UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/779,328	02/13/2004	Umesh Madan	MS1-1851US	5250

22801 7590 08/29/2006

LEE & HAYES PLLC
421 W RIVERSIDE AVENUE SUITE 500
SPOKANE, WA 99201

EXAMINER	
TSAI, SHENG JEN	
ART UNIT	**PAPER NUMBER**
2186	

DATE MAILED: 08/29/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)
	10/779,328	MADAN ET AL.
	Examiner	Art Unit
	Sheng-Jen Tsai	2186

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *13 February 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-38* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-38* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-38 are presented for examination in this application (10,779,328) filed on February 13, 2004.

2. ***Claim Rejections - 35 USC § 102***

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

3. Claims 1-5, 7-8, 11-12, 14-15, 17-20, 22-24, 26-27, 31-33 and 37 are rejected under 35 U.S.C. 102(b) as being anticipated by Schneider (US 5,668,987).

As to claim 1, Schneider discloses **a method** [Database system with Subquery Optimizer (title)], **comprising**:

receiving a request to add a new filter [After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33)] **to a filter table** [figure 3A shows examples of filter tables TABLE T1 (310) and TABLE T2 (320)] **stored in an inverse query engine cache** [figure 2, 260 shows the inverse query engine; figure 3B shows a subquery cache (340) which is part of the query engine];

adding the new filter to the filter table [After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33)];

maintaining the inverse query engine cache at or below a maximum cache size [While the number of items is less than the cache size (step 401), the method

continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

wherein the inverse query engine cache is used exclusively by an inverse query engine to store filters associated therewith [Database system and methods are described for improving execution speed of database queries (e.g., for decision support) by optimizing execution of nested queries or "subqueries," such as are commonly used in client/server database environments. In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract)].

As to claim 2, Schneider teaches that **the method as recited in claim 1, further comprising maintaining the size of the inverse query engine cache between an optimal cache size and the maximum cache size** [In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 3, Schneider teaches that **the method as recited in claim 1, wherein the maintaining further comprises**:

determining if the addition of the new filter to the new filter table increases the cache size above the maximum cache size [figure 4A, step 401, "while items <

cache rows" determines if the size of the items to be added would exceed the size of a cache row]; **and**

removing one or more filters from the filter table if the addition of the new filter causes the cache size to exceed the maximum cache size [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

As to claim 4, Schneider teaches that **the method as recited in claim 3, wherein the determining step further comprises**:

determining a relative size of the new filter [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

assigning a filter weight to the new filter based on the relative filter size [the corresponding weight is the size of the filter item; figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

deriving a cache weight by summing filter weights of all filters in the filter table including the new filter [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

comparing the cache weight to the maximum cache size [After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 5, Schneider teaches that **the method as recited in claim 1, wherein the maintaining further comprises**:

identifying a weight associated with the new filter [the corresponding weight is the size of the filter item; figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

adding the weight associated with the new filter to a cache weight that is the sum of filter weights of filters in the filter table, each filter having a filter weight [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

comparing the cache weight to the maximum cache size [While the number of items is less than the cache size (step 401), the method continues to execute and

maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 7, Schneider teaches that **the method as recited in claim 1, further comprising a least recently used filter in the filter table** [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)]; **and**

wherein the maintaining the inverse query engine cache further comprises removing the least recently used filter from the filter table when a size of the inverse query engine cache reaches the maximum cache size [Finally, an "LRU Depth" of the deepest hit in the cache is also tracked. Since the cache can adjust its size dynamically, it itself has a "depth"--that is, how many rows comprise the cache at a given instance. An "LRU Depth" variable is therefore employed to track how deep (i.e., the deepest row) into the cache there has been a "hit" (column 2, lines 20-40)].

As to claim 8, Schneider teaches that **a system** [Database system with Subquery Optimizer (title)], **comprising**:

an inverse query engine [figure 2, 260 shows the inverse query engine] **configured to test an input against a collection of filters** [figure 3A shows examples of filter tables TABLE T1 (310) and TABLE T2 (320)];

cache associated with the inverse query engine [figure 3B shows a subquery cache (340) which is part of the query engine];

a filter table stored in the cache and containing multiple filters [figure 3A shows examples of filter tables TABLE T1 (310) and TABLE T2 (320)]; **and**

a maintainer configured to maintain a size of the filter table within definite cache bounds [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 11, Schneider teaches that **the system as recited in claim 8, further comprising a trim module configured to remove one or more filters from the filter table when the cache reaches a maximum cache size** [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); Finally, an "LRU Depth" of the deepest hit in the cache is also tracked. Since the cache can adjust its size dynamically, it itself has a "depth"--that is, how many rows comprise the cache at a given instance. An "LRU Depth" variable is therefore employed to track how deep (i.e., the deepest row) into the cache there has been a "hit" (column 2, lines 20-40)].

As to claim 12, Schneider teaches that **the system as recited in claim 11, wherein the trim module is further configured to remove the one or more filters from the filter table until the cache is reduced to an optimal cache size** [In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

As to claim 14, Schneider teaches that **the system as recited in claim 11, wherein**:

a filter weight is associated with each filter in the filter table [the corresponding weight is the size of the filter item; figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

the cache size further comprises a cache weight that is a sum of all filter weights in the filter table [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)];

the maximum cache size further comprises a maximum cache weight [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

wherein the trim module is further configured to deduct a filter weight from the cache weight when a filter associated with the filter weight is removed from the filter table [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a <u>least-recently used (LRU) basis</u> (column 7, lines 26-28); since the weight is the size, whenever a filter is removed from the cache, the size of the cache decreases by the size of the filter, so as the weight].

As to claim 15, Schneider teaches that **the system as recited in claim 14, further comprising a cache weight module configured to assign a filter weight to each filter added to the filter table, each filter weight identifying a relative size of a filter with regard to other filters in the filter table** [since the weight is the size, whenever a filter is added to the cache, the size of the cache increases by the size of the filter, so as the weight].

As to claim 17, Schneider discloses **one or more computer-readable media** [main memory, figure 1A, 102; mass storage, figure 1A, 107] **storing computer-executable instructions** [program code, columns 13~66] **that, when executed on a computer** [central processor, figure 1A, 101], **perform the following steps**:

receiving a request to add a new query to an inverse query engine cache that stores multiple queries, each query having a query size associated therewith [After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its <u>input value(s)/result value pair</u> is <u>added</u> to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); figure 4A, step 401, “while items < cache rows” determines if the size of the items to be added would exceed the size of a cache row];

deriving a cache size that is a sum of query sizes of the queries stored in the inverse query engine [In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving <u>an optimal cache size</u> (abstract)];

determining if the cache size is at greater than or equal to a maximum cache size [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)];

removing one or more queries from the inverse query engine cache if the cache size is greater than or equal to the maximum cache size [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)];

deducting the query size of each query removed from the cache size [since the size of the cache is the sum of the size of all queries in the tables, whenever a query is removed from the cache, the size of the cache decreases by the size of the query];

adding the new query to the inverse query engine cache [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

adding a new query size to the cache size, the new query size identifying a size of the new query added to the inverse query engine cache [since the size of the cache is the sum of the size of all queries in the tables, whenever a query is added to the cache, the size of the cache increases by the size of the query].

As to claim 18, refer to "As to claim 7" presented earlier in this Office Action.

As to claim 19, refer to "As to claim 17" presented earlier in this Office Action.

As to claim 20, Schneider teaches that **the one or more computer-readable media as recited in claim 17, wherein the step of adding the new query size to the cache size is performed before determining if the cache size is greater than or equal to the maximum cache size** [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row; While the number of items is <u>less than the cache size</u> (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 22, refer to "As to claim 4" presented earlier in this Office Action.

As to claim 23, refer to "As to claim 4" presented earlier in this Office Action.

As to claim 24, refer to "As to claim 4" presented earlier in this Office Action.

As to claim 26, refer to "As to claim 6" presented earlier in this Office Action.

As to claim 27, Schneider discloses **a method** [Database system with Subquery Optimizer (title)] **for maintaining an inverse query engine cache** [figure 2, 260 shows the inverse query engine; figure 3B shows a subquery cache (340) which is part of the query engine], **comprising**:

determining when inverse query engine cache usage is approaching a cache usage capacity [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row; While the number of items is <u>less than the cache size</u> (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)];

removing one or more filters from the inverse query engine cache when the cache is approaching the cache capacity until the cache usage is reduced to an optimal cache usage [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract)].

As to claim 31, refer to "As to claim 7" presented earlier in this Office Action.

As to claim 32, Schneider teaches that **an inverse query engine having an integrated cache** [figure 2, 260 shows the inverse query engine; figure 3B shows a subquery cache (340) which is part of the query engine].

As to claim 33, Schneider teaches that **the inverse query engine as recited in claim 32, wherein the cache is bound to a finite size** [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 37, refer to "As to claim 7" presented earlier in this Office Action.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 6, 9-10, 13, 16, 21, 25, 28-30 34-36 and 38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schneider (US 5,668,987), and in view of Minami et al. (US Patent Application Publication 2003/0165160).

As to claims 6 and 9-10, Schneider does not teach **identifying one or more expired filters in the filter table; and removing at least one of the identified expired filters**.

However, Minami et al. teach in their invention "Gigabit Ethernet Adapter" the use of an ARP (Address Resolution Protocol) cache module where entries of the cache may expire [A static ARP cache entry is created in the ARP cache when the internal processor requests the ARP cache module create an ARP cache entry. The internal processor may also create dynamic ARP cache entries. A dynamic ARP cache entry <u>exists for time specified by the user</u> before the ARP cache entry expires, and the ARP cache module removes the cache entry. <u>Expiration time for a dynamic ARP cache entry is typically five to 15 minutes</u>. A static ARP cache entry does not normally expire (paragraph 0256)].

Identifying and removing an expired entry that is no longer needed further enhances the cache <u>replacement</u> strategy beyond the commonly adopted "Least Recently Used" scheme, and makes better utilization of the precious resource of cache capacity.

Therefore, it would have been obvious for one of ordinary skills in the art to recognize the benefits of Identifying and removing an expired entry that is no longer needed, as demonstrated by Minami et al., and to incorporate it into the existing

system disclosed by Schneider, to further enhances the cache replacement strategy beyond the commonly adopted "Least Recently Used" scheme, and makes better utilization of the precious resource of cache capacity.

As to claim 13, Minami et al. teach **that the system as recited in claim 11, wherein the trim module is further configured to determine if a permanent flag in a filter is set and, if the permanent flag is set, to leave the filter in the filter table** [The user may create a static ARP cache entry. <u>A static ARP cache entry</u> is normally <u>permanent</u> and does not expire (paragraph 0265)].

As to claim 21, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 25, refer to "As to claim 13" presented earlier in this Office Action.

As to claim 28, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 29, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 30, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 34, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 35, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 36, refer to "As to claims 6 and 9-10" presented earlier in this Office Action.

As to claim 38, refer to "As to claim 13" presented earlier in this Office Action.

6. Claim 16 is rejected under 35 U.S.C. 103(a) as being unpatentable over Schneider (US 5,668,987), and in view of Baum et al. (US 4,928,239).

As to claim 16, Schneider teaches using a "Least Recently Used" algorithm for replacing cache entries [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)], does not teach **a most recently used list**.

However, "most recently used" is exactly the opposite of "least recently used," thus it is obvious to one of ordinary skills in the art that, regarding replacing a cache entry, a "least recently used" entry should be the first one to be replaced while a "most recently used" entry should be the last one to be replaced.

Further, Baum et al. disclose in their invention "cache memory with Variable Fetch and Replacement Schemes" monitoring the "most recently used" entries for the purpose of cache replacement [according to the LRU replacement scheme, each time one of the blocks in the cache is accessed, it is marked most recently used and the LRU status bits of the others are adjusted accordingly (column 5, lines 5-8)].

Therefore, it would have been obvious for one of ordinary skills in the art to recognize that the "most recently used" is implied by "least recently used" as far as cache entry replacement is concerned, as demonstrated by Minami et al., hence lacking patentable significance.

7. ***Related Prior Art***

The following list of prior art is considered to be pertinent to applicant's invention, but not relied upon for claim analysis conducted above.

- DeMarcken et al., (US Patent Application Publication 2004/0249682), "Filling a Query for Travel Planning."
- Haas et al., (US 6,934,699), "System and Method for Loading a Cache with Query Results."
- Gupta et al., (US 7,035,846), "Method, Computer Programs and Apparatus for caching Directory Queries."
- Jackson, (US Patent Application Publication 2003/0123387), "Device and Method for Filtering Network Traffic."
- Bennett, (US Patent Application Publication 2003/0204664), "Cache with Multiway Steering and Modified Cycle Reuse."
- Fu et al., (US Patent Application Publication 2004/0111519), "Access Network Dynamic Firewall."

Conclusion

8. Claims 1-38 are rejected as explained above.

9. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Sheng-Jen Tsai whose telephone number is 571-272-4244. The examiner can normally be reached on 8:30 - 5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matthew Kim can be reached on 571-272-4182. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Sheng-Jen Tsai
Examiner
Art Unit 2186

August 11, 2006

PIERRE BATAILLE
PRIMARY EXAMINER
8/23/06